International Journal of Foundations of Computer Science
Vol. 17, No. 4 (2006) 885–901

World Scientific
www.worldscientific.com

HYBRID SYSTEM VERIFICATION IS NOT A SINECURE — THE ELECTRONIC THROTTLE CONTROL CASE STUDY*

ANSGAR FEHNKER
National ICT Australia† and University of New South Wales, Sydney, NSW 2052, Australia

BRUCE KROGH
Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Received 2 May 2005
Accepted 2 January 2006
Communicated by Farn Wang

ABSTRACT

Though model checking itself is a fully automated process, verifying correctness of a hybrid system design using model checking is not. This paper describes the necessary steps, and choices to be made, to go from an informal description of the problem to the final verification result for a formal model and requirement. It uses an automotive control system for illustration.

1. Introduction

Hybrid systems are characterized by nontrivial interactions between subsystems with discrete and continuous state variables. These interactions can occur continuously or at discrete instants of time. A typical setting is a digital controller in an analog environment. This interaction makes formal verification of hybrid systems not just tedious, but intrinsically difficult. In recent years the field of hybrid systems has seen significant advances. This research has resulted in a number of tools for model checking of hybrid systems such as Hytech [10], Verishift [17], d/dt [4], CheckMate [2], and PHAVer [6].

These tools and other techniques have been applied to a number of case studies in the domains of automotive control, robotics, avionics, and process control. Despite successful applications of verification tools, it has been questioned if these techniques scale to *real life problems*, i.e. problems with a complexity that can be encountered in industry. The Defense Advanced Projects Resarch Agency (DARPA) program Model Based Integration of Embedded Software (MoBIES) included a set of Open Experimental Platforms (OEPs) to assess the limits of current technology

*This is an extended version of [5]. This work was supported in part by the US Defense Advance Projects Research Agency (DARPA) contract nos. F33615-00-C-1701 and F33615-02-C-4029, US Army Research Office (ARO) contract no. DAAD19-01-1-0485

†National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

for hybrid systems verification. This paper considers the Electronic Throttle Control (ETC) problem of the automotive OEP from the MoBIES program. For the ETC, the problem is: Given a MATLAB Simulink/Stateflow simulation model and an informal description of system and requirements, verify the system as represented by the model satisfies the requirements.

In [13], Henzinger et al. review several case studies performed with the tool HyTech, and obtain criteria to decide when model checking with HyTech is promising. In [19], Rushby describes the use of verification in the design process of critical systems and identifies steps in the design process where formal methods could contribute to the quality of the design. In contrast with this related work we assume the known limitations of hybrid verification, and assume that the decision to use formal verification has been made. This paper uses the ETC case study to illustrate the process that leads from the informal specification to verification with the hybrid systems model checker CheckMate.

We will discuss the following steps that were necessary to go from an informal description of the problem to the final verification result

1. **Construct the mathematical model.** Starting point for verification is more often than not an informal system description, accompanied by a simulation model or implemented code. This information is used to understand the mathematical relationships that govern the system.

2. **Obtain the formal requirements.** As with modelling, there is often more to deriving verification requirements than simply translating given informal requirements.

3. **Build a verification model.** Given the requirements and a mathematical system model one can start building a verification model. Having both, properties and mathematical model, is important for determining what aspect of the system has to be included in the verification model.

4. **Set up the verification problem.** To deal with complexity, it might be necessary to divide the verification problem into sub-problems. A well known approach is assume-guarantee reasoning that introduces a number of tractable verification problems, that, when verified individually, imply the correctness of the requirement [11, 14].

5. **Set up the model checking algorithm.** The last step is to set up the model checking algorithm. This might include choosing a proper size for a hash table, defining a proper exploration order, or, since many hybrid system tools rely on numerical routines, to choose suitable numeric tolerances.

These steps will be part of any verification. In practice there is often no clear distinction between the separate steps, and steps 3 to 5 may be iterated a few times.

This paper is organized as follows. The next section gives a brief description of the hybrid system verification tool CheckMate. CheckMate is able to deal with non-linear hybrid systems, in contrast to tools such as HyTech. This ability enables us to model the ETC directly. Note that hybrid systems with linear differential equations are by convention classified as non-linear hybrid systems. Sections 3 to

7 then describe the verification process from informal specifications to final results. Finally, Section 8 discusses future work that is aimed to support the design and verification process.

2. A Brief Introduction to CheckMate

CheckMate is a model-checker for polyhedral invariant hybrid automata (PIHA) [2], a slightly restricted class of hybrid automata. As hybrid automata, PIHAs have a finite number of control locations. In each location a set of differential equations governs the continuous evolution of the continuous state variables. Transitions between locations occur as soon as switching conditions become true. These switching conditions are defined as conjunctions of linear inequalities. Transitions can also reset the continuous state vector by applying an affine mapping .

The model-checking algorithm of Checkmate partitions the state space, and over-ap-proximates the transition relation using flowpipe approximations. CheckMate then model checks the obtained abstraction against an ACTL specification. ACTL is a subset of CTL (computation tree logic) that states universal properties, that is, properties that are true for all trajectories of the system [3].

A flowpipe is the set of all states that are reachable from a given initial set by continuous evolution. A flowpipe can be viewed as a bundle of trajectories. Checkmate uses polyhedra to over-approximate flowpipes. This has the advantage that intersections of approximations with switching conditions and invariants yield polyhedra. The basic steps are manipulations of polyhedra, computing flowpipe approximations, and model checking the resulting finite-state abstraction.

For a differential equation $\dot{x} = f(x)$, with $x \in \mathbb{R}^n$, let $\varphi(x_0, t)$ be the solution at time t from initial state x_0. Given a set of initial states $X(0) \subset \mathbb{R}^n$, we define a flowpipe segment from t_1 to t_2 as the set $\{x | \exists x_0 \in X(0), t \in [t_1, t_2].\ x = \varphi(x_0, t)\}$. The over-approximation of this segment is computed as follows (illustrated in Figure 1):

1. For the vertices $x_{0_1}, \ldots, x_{0_m}$ of $X(0)$ compute $\varphi(x_{0_i}, t_1)$ and $\varphi(x_{0_i}, t_2)$. CheckMate uses numerical integration to compute these points.

2. Compute a polyhedron that encloses these points. CheckMate computes either convex hulls or oriented hyper-rectangles [21], depending on an option set by user. Later in this paper we discuss implications of this choice. This polyhedron is an initial guess, and does not need to include the complete flowpipe segment.

3. Determine the linear inequalities $c_i\, x \leq d_i$, with $c_i \in \mathbb{R}^{1\times n}$ and $d_i \in \mathbb{R}$, that define the initial polyhedron.

4. Solve for each face of the polyhedron the optimization problem

$$\hat{d}_i = \max_{\substack{x_0 \in X(0) \\ t \in [t_1, t_2]}} c_i\, \varphi(x_0, t) \tag{1}$$

The conjunction of the inequalities $c_i\, x \leq \hat{d}_i$ then defines an over-approximation of the flowpipe segment, i.e. of all points that are reachable from $X(0)$ within interval $[t_1, t_2]$ time.

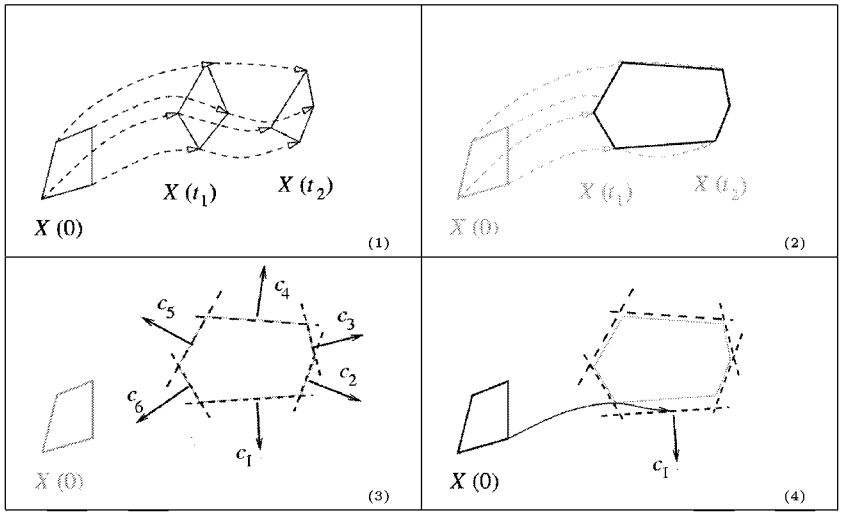

Fig. 1. Steps in the flowpipe approximation are (1) simulating the vertices, (2) enclosing the simulation points in a polyhedron, (3) determining the normals and (4) increasing the size of the polyhedron, until it contains the complete segment.

Extending the flowpipe approximation to PIHAs with parametric differential equations $\dot{x} = f(x,p)$, where p is an unspecified constant parameter, is straightforward. We assume that p is an element of a bounded polyhedron P in $\mathbb{R}^m$. In the first step, we simulate all vertices of $X(0)$ for all vertices of P. In the next two steps, we compute the enclosing polyhedron of the simulation points, as before. The last step includes the parameter in the optimization problem (1):

$$\hat{d}_i = \max_{\substack{x_0 \in X(0) \\ t \in [t_1,t_2] \\ p \in P}} c_i \, \varphi(x_0, t)$$

This defines a polyhedron that includes all states that are reachable from $X(0)$ with parameter values in P and within interval $[t_1, t_2]$ time. Note that while the parameter is assumed constant during continuous evolution, it may change non-deterministically when the analysis evaluates a discrete transition. This includes the case when the parameter remains constant, and the analysis is therefore conservative.

3. Constructing the Mathematical Model

The first step towards verification is to get an understanding of the system behavior. The essential components of the system, the control structure, and the physical laws that govern the behavior must be identified. Information from an informal description may be supplemented by a simulation model.

The mathematical model captures different system characteristics and should reflect the following aspects:

- Dynamics of mechanical, chemical, of biological processes. Typically described by differential equations, partial differential equations, and algebraic constraints.

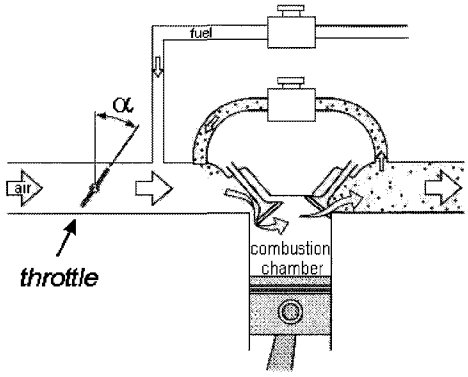

Fig. 2. The angle of the throttle plate determines the air flow, and indirectly the engine speed.

- Switching conditions. Defined by inequalities over state-variable. Even if the system evolves continuously, it may switch autonomously, for example in an elastic collision.
- Time scales. These are determined, for example, by the poles of a linear time-invariant system, or by the sampling rate of a sensor.
- Switching logic. This may be modelled as state chart or as finite state machine. Control logic might also be given as a program, e.g. as relay lader logic or sequential function charts for programmable logic controllers (PLCs).
- Control laws. These are often defined by transfer functions or differential equations; in purely discrete applications, they are encoded completely in the switching logic.
- Communication among components. Communication can be synchronous or asynchronous, with shared events or variables, using message buffers, channels, broadcasting, interrupts, or a combination of those.

If we are given a formal model, rather than an informal description, then the semantics define the mathematical model. In this case this model may be suitable directly for verification. The ETC system, however, was presented as a MATLAB Simulink/Stateflow model, accompanied by an informal description [7, 8]. We use information from simulation and informal description to develop the mathematical model manually.

The ETC system replaces the mechanical link between pedal and throttle plate. Figure 2 depicts the throttle plate as part of the powertrain. The throttle plate angle determines the airflow to the combustion chamber, and (along with the amount of injected fuel and ignition timing) controls the engine torque. The task of the ETC is to control the throttle angle, based on current control mode and human input.

The ETC system comprises a pulse-width modulation (PWM) driver, an actuator (a DC motor), the mechanical system (the throttle and spring), sensors and a controller (Fig. 3). The plant dynamics, i.e., the DC motor and the throttle behavior, are modelled as nonlinear dynamic systems in Simulink. The PWM driver, the

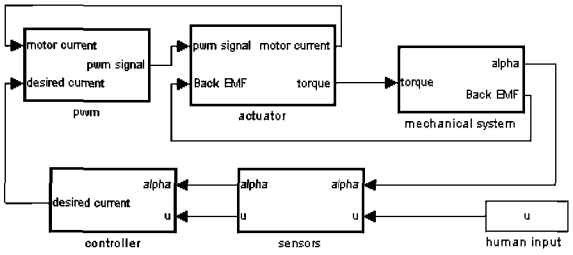

Fig. 3. An automotive throttle control system.

switching logic in the ETC controller, and the scheduler are modelled as Stateflow diagrams. Control laws, such as a sliding mode controller, are modelled as differential equations. Connections between blocks are continuous-time real-valued signals in the Simulink semantics. Switching in the simulation model occurs either by triggered transitions, or by discontinuous Simulink blocks, such as the sign-function block. The remainder of this section describes the mechanical system and the ETC controller in more detail.

The Mechanical System. The behavior of the throttle plate is governed by spring dynamics, Coulomb friction, viscous friction (airflow) and input torque. Variables of the ETC system are the throttle angle α, the angular velocity ω, and the driver input u. Table 1 gives the most important parameters of the ETC model. A changing current in the mechanical system induces an electro-magnetic force (back EMF) that opposes the change. The back EMF is a feedback signal to the actuator. The mechanical system is a second-order nonlinear continuous time-invariant system; the Coulomb friction introduces the nonlinearity in the system. The equations governing the throttle plate dynamics are

$$\dot{\alpha} = \omega \tag{2}$$

$$\dot{\omega} = \frac{1}{J}(-K_f \,\mathrm{sign}(\omega) - \mathrm{K_d}\,\omega - \mathrm{K_s}(\alpha - \alpha_{\mathrm{eq}}) + \mathrm{K_t}\, i_m) \tag{3}$$

The Coulomb friction is proportional to the sign of the angular velocity sign(ω), the viscous friction is proportional to the angular velocity ω, and the force of the spring is proportional to the difference between the actual angle α and the spring equilibrium α_{eq}. Contributing to the angular velocity is the input torque $K_t i_m$. This system is nonlinear, due to the Coulomb friction. Outputs of the mechanical system are the throttle angle α and the back EMF $K_t\omega$.

The ETC controller. The ETC controller has several levels of hierarchy. The top level is a Stateflow diagram, with four normal modes, two failure modes and a startup mode. The human control mode uses a *sliding-mode controller* (described below). All other modes are merely placeholders for undefined implementation details. In the model, the controller delivers just a constant and meaningless output in those modes.

The ETC controller uses a fifth-order filter (with continuous-time poles at -80, -80, -90, -90, -100) to smooth the input from the human driver (the sensor output).

Table 1. Parameters of the ETC dynamics.

$1/J$	Inertia	K_f	Coulomb friction
α_{eq}	Spring equilibrium	K_t	Actuator Gain
λ	Observer Gain	K_d	Damping Gain
η	Controller gain	K_s	Spring constant

The performance of this filter determines in part whether the controller meets its performance requirements. The filter itself can be modelled as linear time-invariant dynamic system of the form

$$\dot{x}_f = A_f x + B_f u \tag{4}$$

$$u_f = C_f x_f + D_f u \tag{5}$$

Sliding-mode controllers are commonly used in control applications, since they are very robust and versatile [22]. Sliding-mode controllers are designed as follows. First, a surface is defined in the state space such that state trajectories on the surface behave as desired, e.g., the state converges to the specified steady state on the surface. Next, for each side of the sliding surface a control law is designed to drive the system to the sliding surface, as illustrated in Figure 4. With these control laws, when the state trajectory hits the sliding surface it stays on the surface and converges to the equilibrium point.

The sliding-mode controller of the ETC has as inputs the filtered input u_f, and the throttle position α. The sliding surface of the ETC is $s = \lambda(\alpha - u_f) + (\omega - \dot{u}_f)$. We say that the system is on the surface if $s = 0$, above the surface if $s > 0$, and below the surface if $s < 0$. The surface is chosen such, that $\omega - \dot{u}_f < 0$ if $\alpha - u_f > 0$, i.e. the difference between the angle and the filtered input angle decreases if the difference is positive. Hence, on the surface the actual angle will converge to the filtered desired angle.

The sliding-mode controller applies the following control law for the desired current:

$$\begin{aligned} i_{desired} = \; & \tfrac{K_s}{K_t}(\alpha - \alpha_{eq}) + \tfrac{K_d}{K_t}\omega + \tfrac{K_f}{K_t}\operatorname{sign}(\omega) \\ & + \tfrac{J}{K_t}\ddot{u}_f - \eta \operatorname{sign}(s) - \lambda \tfrac{J}{K_t}(\omega - \dot{u}_f) \end{aligned} \tag{6}$$

Whether the controller is above or below the surface is encoded as $\operatorname{sign}(s)$. This controller drives the system to the surface. The OEP model uses discrete-time versions of the fifth-order filter and the sliding-mode controller. It takes the numerical derivative of α to obtain ω, and of u_f to obtain $\dot{u}_f$ and $\ddot{u}_f$. The coefficients and parameters in (6) are explained in Table 1.

The model of the controller contains the sliding-mode controller, place holders for the other control modes, the fifth-order filter, blocks that model sampling of input and output, fault detection, delays, a scheduler, in addition to the top level Stateflow model that selects the control mode.

4. Obtaining the Formal Requirements

When handed an informal description of the system requirements, only some will be suitable for formal verification. We can distinguish three types of requirements.

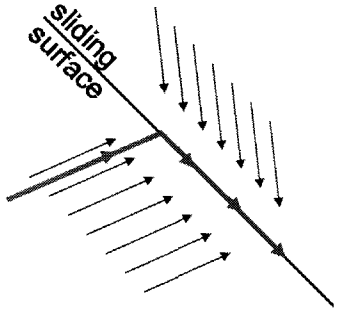

Fig. 4. Illustration of the concept of sliding-mode control.

- Implementation requirements. These requirements impose certain implementation details that can be checked statically. They include requirements on floating point precision, platform, controller, programming language, clock-speed, scheduling policy or input range of sensors. There is no need to examine the dynamic system behavior to check these requirements.

- Requirements on representative behaviors. These requirements define acceptance criteria for particular executions of the system since satisfaction of the requirements can be established by a single run of the simulation model. We refer to these requirements as *simulation requirements*. They often serve as testing scenarios.

- Requirements on classes of behaviors. These requirements define a possibly infinite set of acceptable behaviors, of possibly infinite length. A typical example would be a liveness property such as "Each request is always granted eventually", which is defined for runs of infinite length. We refer to these requirements as *verification requirements*, since they require a formal proof.

The informal description of the ETC lists seven requirements. They include implementation requirements as well as simulation and verification requirements. We will focus on a few of these requirements for illustration.

The requirement that the nominal battery voltage should be 12 VA is a typical implementation requirement. There is no need to use simulation or formal verification, and correctness can be proven by inspection of the relevant parameters.

The rise-time requirement for the ETC is a requirement for representative behaviors. The rise time is defined as "the time required for the throttle plate angle response to a step change in pedal position to rise from 10% of the steady-state value to 90% of the steady-state value". The informal description continues, "The rise time for step changes from closed to fully open is 100ms ... ". The requirement puts bounds on these times, given a particular change in the input signal. Whether this requirement holds can be answered by a single simulation with the test input. The simulation test shows that the rise time requirement is indeed satisfied.

We note that for the rise time requirement – as for most simulation requirements – it is unlikely that the primary interest is how the system reacts to a specific input. This input probably never occurs in reality. The property of interest is

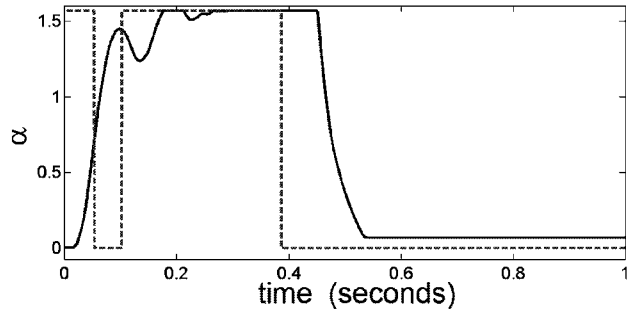

Fig. 5. The throttle angle (solid) in response to an input (dashed) that changes between a wide open throttle, and a fully closed throttle.

the responsiveness of the system. The rise time requirement is just the means to learn about the system's responsiveness. This requirement is widely accepted since the control engineer infers properties of the overall system from simulations with particular inputs.

Another requirement is informally expressed as: "[The] throttle plate shall never hit the stops" [8, p. 10]. This requirement has to hold for any input or operation mode (failure modes are excluded). This requirement is a candidate for formal verification. However, running just a few simulations shows that it is possible to reach the upper bound with a positive velocity, as can be seen in Figure 5, at approximately time 0.2 seconds. Formal verification is therefore not necessary. The counterexample proves that the requirement is not satisfied. The designers of the ETC model confirmed this and other counterexamples, but did not want to relax the requirement since they considered these violations as not significant. The requirements are not just true or false, but certain violations are acceptable within a certain, albeit subjective range.

We developed the following verification requirement for the ETC that can be checked formally: The system will, after a step input, always eventually enter a certain neighborhood of the steady-state, and remain there forever. This property could be expressed in CTL as **AF AG** q, where q is true for states in the neighborhood of the steady-state. This property has an unlimited time horizon, unlike simulation requirements. In addition, we assume that spring constant and spring equilibrium may deviate from their nominal values by up to 20%. Rather than defining infinitely many behaviors for a single system, this requirement defines a single behavior of infinite length for an infinite class of systems. Simulations in contrast require the parameters to be known exactly. In the verification model, these parameters are only known within bounds, and the verification covers all parameters values within these bounds.

5. Obtaining the Verification Model

A limiting factor in hybrid system verification is the number of continuous variables and the number of control locations [13]. Verifying a model with a fifth-order system just to filter the input is already challenging. The ETC system has a few additional characteristics that make verification of the full model impossible. As in

other applications, we do not verify the implementation model but a scaled-down version [19].

A well known technique for scaling down hybrid system models is abstraction [11]. An abstraction preserves the essential behavior of the original system. It is guaranteed that ACTL properties – which includes safety and universally quantified liveness properties – that hold for the abstraction, also holds for the original system. But techniques from system and control theory, such as order-reduction and linearization, can also be useful to obtain proper approximations of the original system.

Considerations when building a verification model are the following.

- The form of the model. The verification model must be in the class of systems that the model checker can handle. Or vice versa, one has to choose a tool that can handle the class of systems.
- The kind of dynamic behavior. Models with nonlinear dynamics are harder to analyze than models with linear dynamics, which are harder to analyze than multi-rate problems.
- The number of continuous variables. Even if the problem has simple dynamics, additional continuous variables add complexity.
- The switching behavior of the system. Even systems with few locations can have undesirable switching behavior. A particular example is Zeno-behavior, i.e. a behavior that exhibits an infinite number of discrete transitions in finite time.

We illustrate the process of obtaining a CheckMate model for the ETC case study. The starting point is the OEP model. The OEP model serves two purposes: It is used for simulation studies, and it is used as a blueprint for implementation. There is limited incentive to be concerned about complexity, since the model is used for simulation rather than verification. As a blueprint for implementation the OEP model contains details such as what task has to run on what platform and under which scheduling policy. On the other hand, when implementation details are unknown the model contains empty subsystems that serve as placeholders for future implementation.

PIHAs are continuous-time models[‡] and can include nonlinear dynamics. However, some nonlinearities can cause numerical problems, and a linear abstraction might be easier to analyze. The number of variables is a concern, too; models with more than 5 continuous variables are typically hard to analyze. Verification for more than 10 continuous variables is in most cases impossible. Furthermore, CheckMate assumes that no two transitions can happen in zero time, which in particular excludes certain Zeno-behavior.

Obtaining a Continuous-Time Model. Since CheckMate models are continuous-time, the discrete-time components of the OEP model must be replaced by appropriate continuous-time components. Discrete-time components in the OEP model are PWM driver, sensors and ETC controller.

[‡]There are extensions of CheckMate that allow for discrete time and sample-data models [15, 20].

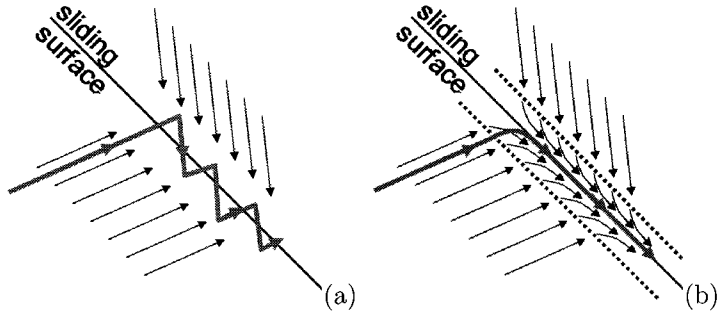

Fig. 6. The left figure illustrates chattering at the sliding surface. The right figure illustrates the effect of a boundary layer.

The Simulink model of the ETC controller has only one mode with meaningful dynamics, the human control mode. We omit in the verification model the other modes, and can omit also the control logic. The filter and sliding-mode controller were designed as continuous-time models, but then discretized to become part of the ETC controller. Hence, we replace them by their continuous-time equivalents. The sampling times of the PWM driver and sensors are a few orders smaller than the time scale of interest (100ms) and these components can be replaced by simple gains.

The sliding-mode controller uses the numeric derivatives of the throttle angle α and of the filtered input u_f. In the continuous-time model we can replace the numeric derivative of α by ω. To deal with the numeric derivatives of u_f, we observe that the requirement is formulated for step inputs. We can assume that $\dot{u} = 0$ (except for a finite number of points), and substitute $\dot{u}_f$ and $\ddot{u}_f$ in (6), using (4) and (5), as follows:

$$\dot{u}_f = C_f A_f x + C_f B_f u \tag{7}$$

$$\ddot{u}_f = C_f A_f^2 x + C_f A_f B_f u \tag{8}$$

Resolving Zenoness. CheckMate assumes, as most hybrid model checkers do, that all behaviors are non-Zeno. The sliding-mode controller, in contrast, intentionally drives the system to a surface where infinite, and even uncountable switching occurs (in the continuous-time realization). If we use a fixed-step integration routine the solution will start chattering, which may lead to unreliable results (see Figure 6(a)). If we use a variable-step integration routine the procedure tends to get stuck on the sliding surface.

To resolve this problem, we define a *boundary layer* (or ϵ-*neighborhood*) around the sliding surface. We apply the sliding-mode controller outside of this ϵ-neighborhood, and replace it inside by a control law that is continuous and drives the system to the surface. The controller is thus equivalent to the original controller outside the boundary layer, and there is a steep but continuous transition from one sliding mode to the other. On the sliding surface the control law is equal to the so-called equivalent controller. Figure 6 (b) depicts the basic idea of a boundary layer. The boundary layer leads to a numerically well-conditioned, non-Zeno, and

close approximation of the ideal sliding-mode behavior. Boundary layers are a very common approach used in physical systems to mitigate the physical stress by chattering that can lead to mechanical damage. A more thorough discussion of reachability analysis for sliding-mode controllers can be found in [16].

Modelling Nonlinearities. The mechanical system describes the dynamics of the throttle plate. This second-order system is nonlinear due to coulomb friction. We have to decide whether to include this nonlinearity and nonlinearities caused by saturation (actuator) and sliding-mode control as different modes, or as nonlinearities. One extreme choice would be to model the ETC as nonlinear hybrid system with a single mode. The other extreme choice would be to model it with linear dynamics, which results in 18 modes for the ETC problem.

If we put all behavior in a single nonlinear differential equation, the flowpipe-approximation gets worse and computationally more expensive when the vector field changes abruptly, e.g. when the system switches between sliding modes.

If we model the system with many modes but with linear dynamics, it will result in a lot of switching. Each time the analysis algorithm encounters switching between modes it uses over-approximations of previous steps, and over-approximation errors may proliferate. Thus, many modes lead also to an increased over-approximation error. Hence, we must decide which approach works best for which non-linearity. We have chosen to model the Coulomb friction and saturation as nonlinearities, to reduce the number of modes, and to model the sliding-mode controller as different modes, to avoid over-approximation errors due to sudden changes of the vector field. This decision was made after running a number of experiments with different models.

Reducing the Order. The ETC uses a fifth-order filter to smooth the input from the human driver. This means that the filter alone has more than twice as many state variables than in the rest of the system. Since verification of hybrid systems becomes more difficult with each additional dimension, we reduce the order of the filter. We obtain a reduced-order filter using the model-reduction capabilities of MATLAB's system identification toolbox. The combined dynamics of plant and reduced filter result in a fourth-order system with nonlinear dynamics. For a more thorough discussion of order-reduction for hybrid system verification see [9].

6. Setting Up the Problem

This section addresses the problem that the requirement cannot be verified directly, due to the size of the verification problem. A common approach is to decompose system and property into smaller problems [12, 14, 18]. We illustrate this idea, by decomposing the liveness property for the ETC into a series of properties, each of which can be verified with CheckMate.

The property that we verify is the following. Given that the system is in steady-state with throttle angle $\alpha = 0$, assume a step change in the desired angle to 89.8 degrees (which is the maximal input; the input has a safety margin of 0.2 degrees) at time 0. Verify that the angle will always eventually reach a 2% neighborhood of the desired angle, and remain there forever. We furthermore assume that the spring constant and spring equilibrium may deviate from their nominal values by 20%. This means that they may take any value in this range.

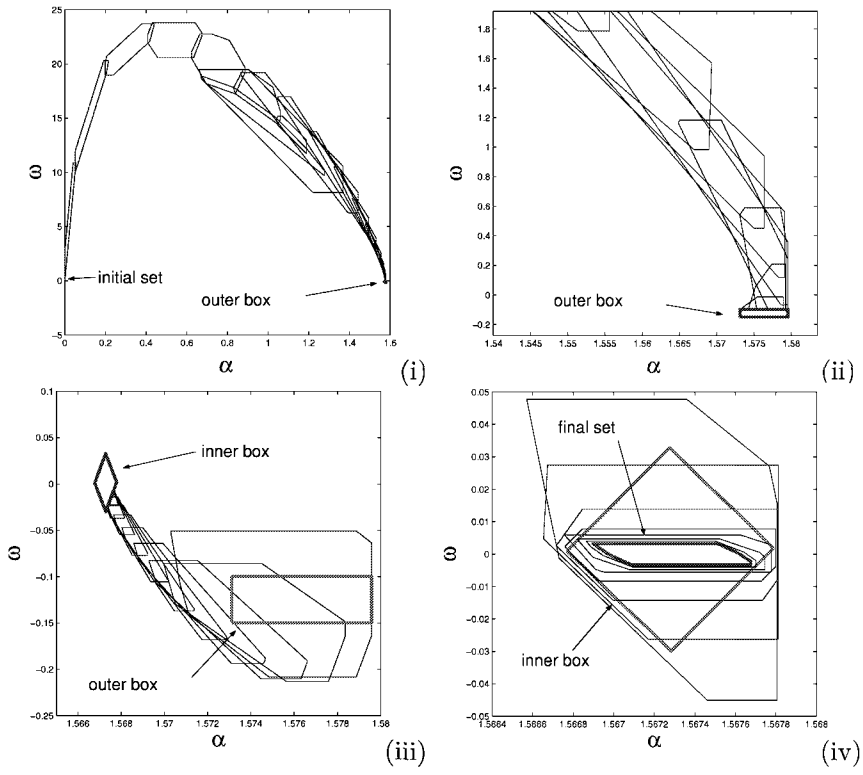

Fig. 7. Figure (i) and (ii) depict the flowpipe approximation for the transient phase. Figure (ii) is a close up of Figure (i). Figure (iii) depicts the flowpipe approximation in the neighborhood of the steady-state value. Finally, Figure (iv) shows that the states that start in the inner box will eventually return to this set. Note that these figures show projections of the flowpipe segments onto throttle angle α and angular velocity ω.

We define a cascade of subproblems to show that the system behaves as desired. For each of the stages we use a variant of the basic CheckMate model. For each stage we define a different initial and target set. The target set of one stage is then the initial set of the next stage.

Transient phase. The first stage of the verification cascade deals with the transient phase when the throttle angle changes quickly in response to the step input. We show that all trajectories that start from the initial set – in this case the origin – hit the first target set, the so called *outer box*. Figure 7 (i) and (ii) depict projections of the flowpipe approximations that show that all trajectories do indeed reach the outer box. The model checker verifies furthermore that the system will always reach this set eventually.

Regulation phase. The next stage is to show that all trajectories that start in the *outer box* will eventually reach the *inner box*. We use the same model as for the transient phase, but of course with the outer box of the transient phase as initial set, and the inner box as target set. Figure 7(iii) shows that when the system

starts in the outer box, all trajectories converge quickly to a neighborhood of the steady-state. No segment of the flowpipe approximation violates the 2% bound. This guarantees that once the system enters the outer box, the inner box will be reached without violating the 2% bound.

Asymptotic behavior. As the last step we show that the *inner box*, a neighborhood of the steady-state value, maps onto itself in a finite number of steps. CheckMate finds a flowpipe segment that is completely contained in the inner box, i.e. the initial set of this stage. This means all trajectories that start in the inner box, return to this set. None of the computed flowpipe segments of the over-approximation violates the 2% threshold.

Figures 7(iv) depicts the result of the final verification step. Note that it is not sufficient to show that some flow pipe segment is contained in another, since they are over-approximations. We cannot assume that all states in a segment are actually reachable. But if some segment is inside the initial set we know that this set is recurrent. All states that can be reached in a certain time interval from the initial set will be contained in this segment, and thus also in the initial set. This completes the verification.

7. Setting Up the Verification Algorithm

The previous section presented the verification results. Getting the verification results is not only a matter of defining subproblems and corresponding models – which is some work by itself – getting the verification to run requires also a fair amount of tweaking the model checker. For finite-state model checkers this might entail choosing a proper order of the variables, for other model checkers it might entail to find a proper size for the hash table. To give an impression of the kind of choices that must be made for CheckMate, we elaborate on the choice when to use convex hulls and when to use oriented rectangular hulls in the approximation.

This choice makes a difference in two different steps of the algorithm. As mentioned before, CheckMate obtains an initial approximation of the flowpipe segment by computing a polyhedron that encloses the simulation points (step 2, page 883). The convex hull of these points is by definition the smallest polyhedron that contains all points. Using the convex hull in this step has the advantage that the over-approximation error is small. A drawback is that the convex hull will also yield polyhedra with a lot of faces. Each additional face leads to one additional optimization problem in the last step of the flowpipe approximation procedure.

CheckMate offers as an alternative to use the so called oriented rectangular hull (ORH) routine [21]. The ORH routine chooses an oriented hyper-rectangle that keeps the over-approximation error small and limits at the same time the number of faces. For the ETC model with four state variables, the ORH will result in a polyhedron with exactly eight faces. If we use the convex hull approximation instead, CheckMate computes polyhedra with up to 119 faces before it gets stuck. Using the ORH solves this problem, and all segments of the approximation can be computed.

Another point in the procedure where the choice between the convex hull and the ORH routine matters, is when a set of reachable states triggers a discrete transition. To compute the successor, CheckMate intersects each segment of the

flowpipe approximation with the switching condition. If more than one segment intersects with a switching condition, the verification algorithm proceeds with an over-approximation of the union of these intersections. This over-approximation can either be the convex hull or the ORH of those sets. For this case study we found that the results of the ORH are too conservative. The over-approximation error soon becomes too large.

To summarize, getting the verification to run requires a proper setup of the verification algorithm. We use, for example, the ORH routine to compute the polyhedra of the flowpipe approximation, and the less conservative convex hull routine to compute the over-approximation of the intersections with switching conditions. Similar choices had to be made to find the proper integration routine, and to chose parameters for numerical integration and optimization routines.

8. Discussion

The starting point for the work presented in this paper was the OEP model and an informal description of the ETC system. The first step towards verification was to construct the underlying mathematical model. The OEP model was useful since it already provided information about the main components. When a such a model is not present in the beginning, building a simulation model can help significantly to understand the problem.

The second step towards the verification was to formulate the requirements of the system. In our case we had an informal description to start from. We found that none of the given requirements was suitable for verification. Most of the requirements were simulation requirements that could be checked by running a simulation, or implementation requirements that could be checked by inspection. For others we easily found counterexamples, and verification was not necessary either.

There is no need to use verification to check simulation and implementation requirements. Verification methods should not be used for the sake of verification. Even more so, since verification can be a tedious effort, even if supported by model checking tools, as this case study shows. Verification should be used if the requirements are formulated for parametric models, uncertain initial conditions, or non-deterministic models. In that case formal verification can be valuable and complementary to simulation-based methods, and worth the effort. We defined a liveness property for the system that captures the sprit of the simulation scenarios. It also illustrates the added value of verification, since these properties deal with an infinite set of behaviors of infinite length and cannot be proven by simulation.

The mathematical model and the liveness requirement were the basis of the verification model. Building the verification model involved simulation of various models. We took into account that we needed a continuous-time model, with a small number of continuous variables, and dynamics that are numerically well-conditioned. The final result was a fourth-order hybrid system with nonlinear dynamics.

Given the verification model we could not verify the requirement directly, but decomposed the problem into smaller problems. For the ETC case study it was sufficient to define a series of three problems that were solved by CheckMate. Finally, we had to find suitable verification parameters, which required several experiments with different settings.

The translation from the Simulink/Stateflow model was performed manually in this paper. Tools that translate Simulink/Stateflow models automatically are unfortunately not yet capable to replace this manual translation. The original ETC model was not suitable for verification for a number of reasons discussed in this paper, and any direct translation would lead to a model with the very same problems. Related work in the Mobies project on a translator from Simulink/Stateflow to HSIF, a hybrid automata based interchange format [1], tried to translate the scaled down version described in this paper. But even then the result was not suitable for verification. Currently, automatic translation typically increases the size of the problem rather than reducing it. For the foreseeable future some user interaction remains necessary to make appropriate choices and simplifications. We expect that eventually automatic translations will be used to replace most of these steps, just like Matlab's systems identification was used in this paper to reduce the size of the filter model.

An interesting observation was that the counterexamples that were found in the early stages, were not considered to be significant by the designers. There were subjective acceptance criteria, which are hard to formalize. The solution to this problem are not probabilistic approaches, or multi-valued logics. Those are formal methods, too, and require also a mathematical precise formulation of the requirements. The issue is that there are humans involved in the design process, who determine whether a counterexample is acceptable based on experience and domain knowledge.

Given our experience from the ETC case study, future research should focus on supporting the process described in this paper. Hybrid systems verification will in the foreseeable future not become a completely automated process. There is a lot of work currently focussing on automating and supporting particular steps, but little that aims to support the complete process. Tool support can be useful to guide and assist the designer throughout the process that leads from informal description to verification result. At the same time it can help to make this process transparent, such that the steps and choices can be re-evaluated at a later stage.

The Simulink/stateflow model and the CheckMate models have a completely different structure – the OEP model is decomposed in plant, controller actuator and sensor, whereas the Checkmate model separates the continuous and discrete part. Those models are related by a number of intermediate models, obtained by abstraction or refinement, as well by approximation, discretization, order-reduction or by automatic or manual translation. Keeping track of the different models, and why and how their were obtained, can help to assess why a system was found to comply with its requirements. This would take into account that formal verification of a hybrid system is not just algorithmic, but a creative process.

References

1. A. Agrawal, G Simon, and G. Karsai. Semantic translation of Simulink/Stateflow models to hybrid automata using graph transformations. *Electr. Notes Theor. Comput. Sci.*, 109, 2004.
2. A. Chutinan and B.H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *HSCC 99*, LNCS 1569. Springer Verlag, 1999.

3. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking.* MIT Press, 2000.
4. Thao Dang. *Vérification et synthèse des systémes hybrides.* PhD thesis, Verimag, Grenoble, 1999.
5. A. Fehnker and B.H. Krogh. Hybrid system verification is not a sinecure: The electronic throttle control case study. In *ATVA 2004, Taipei, Taiwan, ROC.*, LNCS 3299, 2004.
6. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *HSCC 01*, LNCS 2289, pages 258–273. Springer-Verlag, 2005.
7. P. Griffiths. Embedded software control design for an electronic throttle body. Master's thesis, UC Berkeley, 2002.
8. Automotive OEP Group. Electronic throttle control – end-to-end challenge problem for automotive mid-term experiment. August 2001, Available at: http://vehicle.me.berkeley.edu/mobies/papers/etc_challenge_problem_8_6_01.doc.
9. Z. Han and B.H. Krogh. Using reduced-order models in reachability analysis of hybrid systems. In *ACC 2004*, 2004.
10. T.A. Henzinger, P. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *STTT*, 1(1-2):110–122, 1997.
11. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):540–554, 1998.
12. T.A. Henzinger, M. Minea, and V. Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. In *HSCC 01*, LNCS 2034. Springer-Verlag, 2001.
13. T.A. Henzinger, J. Preussig, and H. Wong-Toi. Some lessons from the HyTech experience. In *40th CDC*, pages 2887–2892. IEEE Press, 2001.
14. R. Huuck, B. Lukoschus, G. Frehse, and S. Engell. Compositional verification of continuous-discrete systems. In *Modelling, Analysis and Design of Hybrid Systems*, LNCIS 279. Springer, 2002.
15. J. Kapinski and B.H. Krogh. A new tool for verifying computer controlled systems. In *Computer-Aided Control System Design*, 2002.
16. J. Kapinski and B.H. Krogh. Verifying asymptotic bounds for discrete-time sliding mode systems with disturbance inputs. In *Proc of ACC*, 2004.
17. A. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In N. Lynch and B. Krogh, editors, *HSCC*, LNCS 1790, pages 203–213. Springer, 2000.
18. N.A. Lynch, R. Segala, and F.W. Vaandrager. Hybrid I/O automata revisited. In *HSCC 01*, LNCS 2034. Springer-Verlag, 2001.
19. J. Rushby. Formal methods and their role in the certification of critical systems. Technical Report SRI-CSL-95-1, SRI, Menlo Park, CA, 1995.
20. B.I. Silva and B.H. Krogh. Modeling and verification of hybrid system with clocked and unclocked events. In *40th CDC*, 2001.
21. O. Strusberg and B. Krogh. On efficient representation and computation of reachable sets for hybrid systems. In *HSCC'2003*, LNCS 2289. Springer, 2003.
22. V. Utkin. *Variable structure systems with sliding modes.* AC-22(2):212-222. IEEE Transactions on Automatic Control, 1977.